

Osa V: Arendus ja töövood — täisspikker

Tase: Edasijõudnu

Git, GitHub ja töövoog

Eesmärk: Git hoiab muutuste ajalugu; erista tööpuud, vaheala ja versioonikirjet ning vaata muudatused enne üle.

Põhikujud

- `git status` — vaata seis
- `git diff` — muutused tööpuus
- `git diff --cached` — muutused vahealal
- `git add fail.txt` — lisa vahealale
- `git commit -m '...'` — loo versioonikirje
- `git switch -c parandus` — loo haru
- `git pull --ff-only` — uuenda puhtalt
- `git push -u origin parandus` — saada haru
- `git restore --staged fail.txt` — eemalda vahealalt

Olulisemad lipud, märgid ja kiirnupud

- `HEAD` — praegune tipp
- `main` — põhirida
- `origin` — kaugrepo
- `vaheala` — järgmise versioonikirje sisu

Pane tähele: Uus jälgimata fail ei ilmu tavalisse `git diff` vaatesse; pärast `git add` kontrolli esimese versioonikirje sisu käsuga `git diff --cached`.

Pythoni venv ja eraldatud keskkonnad

Eesmärk: venv hoiab ühe projekti Pythoni paketid eraldi; kõige olulisem on, et terminal, pip ja IDE näeksid sama keskkonda.

Põhikujud

- `python3 -m venv .venv` — loo keskkond
- `source .venv/bin/activate` — aktiveeri shellis
- `python -m pip install -U pip` — uuenda pip
- `python -m pip install requests` — paigalda pakett
- `python -m pip list` — vaata pakke
- `command -v python` — kontrolli tõlgendit
- `deactivate` — välju keskkonnast

Olulised märgid

- `(.venv)` — aktiivne keskkond
- `.venv/` — projekti sees
- `python -m pip` — kindlam kui pip
- `Docker` — terve keskkond

Pane tähele: Kui pip install läks valesse kohta, kontrolli kohe command `-v` python ja command `-v` pip; enamasti on probleem just vales aktiivses keskkonnas.

Docker'i alused

Eesmärk: Konteinerkuju (image) on käivitusmall; konteiner võib töötada või olla peatatud. Konteineri eemaldamisel kaob selle kirjutatav kiht, mitte seotud hostifailid ega püsivad andmeköited.

Põhikäsud

- `docker --version` — kontrolli olemasolu
- `docker run --rm alpine echo tere` — ohutu esimene käivitus
- `docker build -t rakendus .` — koosta konteinerkuju
- `docker run --rm rakendus` — käivita konteiner
- `docker ps` — töötavad konteinerid
- `docker logs -f nimi` — jälgi logi
- `docker exec -it nimi sh` — sisene töötavasse konteinerisse
- `docker compose up -d` — teenused taustale
- `docker compose down` — peata ja eemalda konteinerid ning võrk

Olulisemad lipud, märgid ja kiirpud

- `image` — konteinerkuju ehk käivitusmall
- `konteiner` — käivitatud või peatatud eksemplar
- `Dockerfile` — konteinerkuju retsept
- `compose.yaml` — mitu teenust
- `volume` — püsiv andmeköide
- `registry` — konteinerkujude hoidla
- `--rm` — kustuta lühikatsse konteiner pärast lõppu
- `-p 127.0.0.1:8000:8000` — docker run: avalda port ainult oma arvutile
- `--mount type=bind,src="$PWD",dst=/app` — jaga praegune kataloog konteinerisse
- `-w /app` — määra konteineri töökataloog
- `docker compose down -v` — peatab ja kustutab ka andmeköited

Pane tähele: Kui konteiner ei tööta, vaata enne `docker logs`; ära seo tundmatu konteinerkujuga oma kodukataloogi ega salajasi faile.

IDE-d ja arenduskeskkonnad

Eesmärk: heas arenduskeskkonnas osutavad terminal, Git, tõlgendaja ja vajaduse korral konteiner samale projektile.

Kontrollid ja valikud

- `python3 --version` — kontrolli Pythonit
- `python3 -m pip --version` — kontrolli pip-i
- `git --version` — kontrolli Giti
- `docker --version` — kontrolli Dockerit
- `node --version` — kontrolli Node'i
- `npm --version` — kontrolli npm-i

Olulised mõisted

- tõlgendaja — koodi tegelik käivitaja
- sisseehitatud terminal — sama projektivaade
- Remote SSH — tööta kaugmasinas
- devcontainer — IDE konteineris

Pane tähele: Kui IDE-s miski ei tööta, kontrolli kõigepealt, kas IDE kasutab sama Pythonit, Node'i või konteinerit kui terminal.

Andmeteaduse eelteadmised käsurea vaates

Eesmärk: andmeteaduse stardis ei piisa ühest tööriistast; vaja on korraga failivormingute tunnetust, SQL-i mõtteviisi, programmeerimist ja statistilist mõtlemist.

Põhikujud

- `fail -> head/less -> column/jq -> sqlite3 -> python3` — alusta väikese vaatusega
- Pythoni `venv` — projekti töölaud
- CSV, JSON ja XML — andmete kuju
- SQLite ja Python — esimene andmebaas
- Teksti teisendamine — väiksed filtrid
- Vood ja tabelid — koondamine

Põhiteljed

- programmeerimine — korduv töö loogikaks
- SQL — küsi ja seo andmeid
- vormingud — CSV, JSON, XML
- statistika — anna tähendus

Pane tähele: Ära mine otse mudeli või märkmiku (notebook) juurde enne, kui oled vähemalt korra kontrollinud, mis kujul andmed sul tegelikult on.

CSV, JSON ja XML käsuraal

Eesmärk: vormingu tüüp ütleb, milline tööriist mõistab andmeid kõige loomulikumalt: tabelit, puud või märgendistruktuuri.

Põhikujud

- `head fail.csv` — vaata CSV algust
- `column -s, -t < fail.csv` — joonda lihtne CSV
- `cut -d, -f1 fail.csv` — võta üks veerg
- `python3 -m json.tool fail.json` — vorminda JSON
- `jq '.students[].name' fail.json` — vali JSON välju
- `xmllint --format fail.xml` — vorminda XML

Olulised mõisted

- CSV — lihtne tabel
- JSON — objektide puu
- XML — märgendipuu
- jq — JSON filtriks

Pane tähele: Kui CSV-s on jutumärgid, komad välja sees või reavahetused, ei pruugi `cut -d`, enam piisata; siis mine pigem Pythonisse või spetsiaalsema parseri juurde.

Andmebaasi algus: sqlite ja Python

Eesmärk: SQLite on sild tekstifailide ja päris andmebaasimõtte vahel: failina lihtne, SQL-i mõttes siiski relatsiooniline andmebaas.

Põhikujud

- `sqlite3 andmed.db` — ava või loo fail
- `sqlite3 andmed.db '.tables'` — vaata tabeleid
- `sqlite3 andmed.db '.schema'` — vaata struktuuri
- `sqlite3 andmed.db 'select * from students limit 5;'` — piilu ridu
- `sqlite3 andmed.db 'select city, count(*) from students group by city;'` — koonda read
- `sqlite3 andmed.db 'select s.name, r.score from results r join students s on s.id = r.student_id;'` — ühenda tabelid
- `python3 naide.py` — kasuta Pythonist

Olulised mõisted

- `primary key` — rea unikaalne id
- `foreign key` — viide teise tabelisse
- `JOIN` — seo tabelid

- `GROUP BY` — koonda read

Pane tähele: Ära tee `JOIN`-i ainult käsu pärast; enne joonista enda jaoks välja, milline väli viitab millisele tabelile.

Kompileerimine ja käivitamine: shell, Python, C, C++, Go, Rust, Java

Eesmärk: Erista lähtekoodi, koostatud väljundit ja selle käivitajat. Vali üks paigaldatud tööriistaga keel.

Käsud ja näited

- `sh hello.sh` — tõlgenda POSIX-shelliskript
- `python3 hello.py` — käivita Pythoni lähtefail
- `cc hello.c -o hello-c` — kompileeri C lähtefail käivitatavaks failiks
- `./hello-c` — käivita äsja koostatud C programm
- `c++ hello.cpp -o hello-cpp` — kompileeri C++
- `go build -o hello-go hello.go` — koosta Go binaarfail
- `cargo run` — Rust: koosta ja käivita Cargo projekt
- `javac Hello.java` — Java: loo baitkoodifail `Hello.class`
- `java Hello` — käivita klass JVM-is, ilma `.class`-laiendita

Pane tähele: `./fail` vajab sinu süsteemile sobivat käivitatavat faili; lähtekoodi ega teise platvormi binaari ei saa nii lihtsalt käivitada.

LaTeX käsurealt

Eesmärk: LaTeX-i mõte on lihtne: lähtetekst jääb tekstifailiks, aga buildist sünnib väga täpselt juhitud PDF.

Põhikujud

- `pdflatex tere.tex` — kompileeri PDF
- `xelatex tere.tex` — parem Unicode tugi
- `latexmk -pdf tere.tex` — korda vajalikud käigud
- `latexmk -xelatex tere.tex` — xelatex automaatselt
- `ls tere.*` — vaata abifaile
- `open tere.pdf` — ava tulemus

Olulised failid

- `.tex` — lähtefail
- `.pdf` — tulemus
- `.aux` — ristviidete abi
- `.log` — kompileerimislogi

Pane tähele: Kui ristviited või sisukord ei ilmu kohe õigesti, ei tähenda see tingimata viga; LaTeX vajab sageli rohkem kui üht kompileerimiskäiku.